Jack Girard
ECE 637
Lab 1: Image Filtering

1. C Programming

No Deliverables

2. Displaying and Exporting Images in Python

No Deliverables

3. FIR Low Pass Filter

1. Derivation of $H(e^{j\mu}, e^{j\nu})$: *See Next Page*

Jack Girard
ECE637
Lab 1 Work

$$h(m,n) = \begin{cases} 1/81 & \text{for } |m| \le 4 \text{ and } |n| \le 4 \\ 0 & \text{otherwise} \end{cases}$$

Identity Used:

$$sin(x) = \frac{e^{ix} - e^{-ix}}{2i}$$

$$cos(x) = \frac{e^{ix} + e^{-ix}}{2}$$

Apply formula for DSFT & Simplify:

$$H(e^{j\mu}, e^{j\nu}) = \sum_{n=-\infty}^{\infty} \sum_{m=-\infty}^{\infty} h(m,n) e^{-j(\mu m + \nu n)}$$

$$= \sum_{n=-4}^{4} \sum_{m=-4}^{4} \frac{1}{81} e^{-j(\mu m + \nu n)}$$

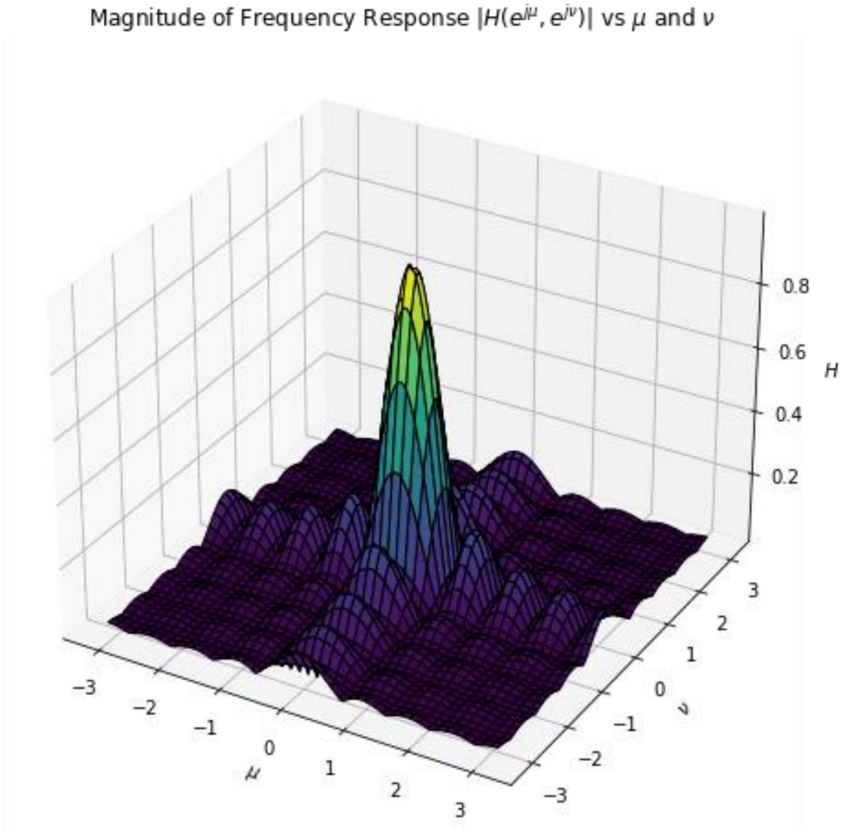
$$= \frac{1}{81} \sum_{n=-4}^{4} e^{-j\mu m} \sum_{m=-4}^{4} e^{-j\nu n}$$

$$= \frac{1}{81} \Big[\left(1 + e^{-4\mu} + e^{4\mu} + e^{-3\mu} + e^{3\mu} + e^{-2\mu} + e^{2\mu} + e^{-\mu} + e^{\mu}\right) \ldots$$
$$\left(1 + e^{-4\nu} + e^{4\nu} + e^{-3\nu} + e^{3\nu} + e^{-2\nu} + e^{2\nu} + e^{-\nu} + e^{\nu}\right) \Big]$$

$$= \frac{1}{81} \Big[\left(1 + 2\cos(4\mu) + 2\cos(3\mu) + 2\cos(2\mu) + 2\cos(\mu)\right) \ldots$$
$$\left(1 + 2\cos(4\nu) + 2\cos(3\nu) + 2\cos(2\nu) + 2\cos(\nu)\right) \Big]$$

$$= \boxed{\frac{1}{81} \left[\left(1 + 2\sum_{k=1}^{4} \cos(k\mu)\right)\left(1 + 2\sum_{l=1}^{4} \cos(l\nu)\right) \right]}$$

2. Plot of $\left|H\left(e^{j\mu}, e^{j\nu}\right)\right|$



3. Color image *img03.tif*

4. The filtered image *color.tif*

5. Listing of C Code: *See Next Page*

```c

#include <math.h>
#include "tiff.h"
#include "allocate.h"
#include "randlib.h"
#include "typeutil.h"

void error(char *name);
// initialize limitIntensity function
int limitIntensity(double value);
// initialize applyFilter function
void applyFilter(struct TIFF_img* output_img, struct TIFF_img* input_img, double** usedFilter, int PSF_dim);

int main (int argc, char **argv)
{
    FILE* fp;
    struct TIFF_img input_img, color_img;

    if ( argc != 2 ) error( argv[0] );

    /* open image file */
    if ( ( fp = fopen(argv[1], "rb") ) == NULL) {
        fprintf( stderr, "cannot open file %s\n", argv[1] );
        exit( 1 );
    }

    /* read image */
    if ( read_TIFF( fp, &input_img ) ) {
        fprintf( stderr, "error reading file %s\n", argv[1] );
        exit( 1 );
    }

    /* close image file */
    fclose( fp );

    /* check the type of image data */
    if ( input_img.TIFF_type != 'c' ) {
        fprintf( stderr, "error:  image must be 24-bit color\n" );
        exit( 1 );
    }

    // declare 1D array of double pointers filter
    double* filter[9];

    // iterate through 1D array filter and allocate enough memory for 9 doubles each
    for (int i = 0; i < 9; i++) {
        filter[i] = malloc(sizeof(double) * 9);
```

```c
    }
    // populate filter array according to h(m,n)
    for (int i = 0; i < 9; i++) {
        for (int j = 0; j < 9; j++) {
            filter[i][j] = 1.0 / 81.0;
        }
    }

    /* set up structure for output color image */
    /* Note that the type is 'c' rather than 'g' */
    get_TIFF( &color_img, input_img.height, input_img.width, 'c' );

    // declare and initialize integer to store the dimension of the point spread function
    int PSF_dim = 9;

    // apply filter using applyFilter function as defined below main
    applyFilter(&color_img, &input_img, filter, PSF_dim);

    /* open color image file */
    if ( ( fp = fopen( "color.tif" , "wb" ) ) == NULL) {
        fprintf( stderr, "cannot open file color.tif\n" );
        exit( 1 );
    }

    /* write color image */
    if ( write_TIFF( fp, &color_img ) ) {
        fprintf( stderr, "error writing TIFF file %s\n", argv[2] );
        exit( 1 );
    }

    /* close color image file */
    fclose( fp );

    /* de-allocate space which was used for the images */
    free_TIFF( &(input_img) );
    free_TIFF( &(color_img) );

    return(0);
}

void error(char* name)
{
    printf("usage:  %s  image.tiff \n\n", name);
    printf("this program reads in a 24-bit color TIFF image.\n");
    printf("It then horizontally filters the green component, adds noise, \n");
    printf("and writes out the result as an 8-bit image\n");
    printf("with the name 'green.tiff'.\n");
```

```c
    printf("It also generates an 8-bit color image,\n");
    printf("that swaps red and green components from the input image");
    exit(1);
}

// limitIntensity function definition
int limitIntensity(double inputValue) {
    // declare an integer variable newValue and initialize it to zero
    int newValue = 0;
    // if input value parameter is less than zero, assign new value to 0
    if (inputValue < 0) {
        newValue = 0;
    }
    // if input value parameter is greater than 255, assign new value to 255
    else if (inputValue > 255) {
        newValue = 255;
    }
    // otherwise, assign new value to the input value parameter re-cast as an integer
    else {
        newValue = (int)inputValue;
    }
    return newValue;
}

// applyFilter function definition
void applyFilter(struct TIFF_img* output_img, struct TIFF_img* input_img, double** usedFilter, int PSF_dim) {
    // declare and define N - the dimension of the point spread function (PSF)
    int N = (PSF_dim - 1) / 2;
    // declare and define image height and width based on input image TIFF struct methods
    int img_height = input_img->height;
    int img_width = input_img->width;
    // declare doubles to store red, green, and blue value for each pixel
    double redPlane, greenPlane, bluePlane;
    // declare PSF variables
    int m, n;
    // declare variables to store current location within PSF
    int a, b;
    // for each pixel:
    for (int i = 0; i < img_height; i++) {
        for (int j = 0; j < img_width; j++) {
            // initialize RGB values to zero
            redPlane = 0.0;
            greenPlane = 0.0;
            bluePlane = 0.0;
```

```
            // for each pixel in the PSF (9*9 in this case)
            for (m = -N; m <= N; m++) {
                for (n = -N; n <= N; n++) {
                    // assign a and b to current PSF matrix location
                    a = i - m;
                    b = j - n;
                    // if a and b are within the image boundaries
                    if (a >= 0 && a < img_height && b >= 0 && b < img_width) {
                        // apply filter by summing across PSF according to difference equation for 2D filters
                        redPlane += usedFilter[m + N][n + N] * input_img->color[0][a][b];
                        greenPlane += usedFilter[m + N][n + N] * input_img->color[1][a][b];
                        bluePlane += usedFilter[m + N][n + N] * input_img->color[2][a][b];
                    }
                }
            }
            // populate output image method for color after calling limitIntensity function to ensure acceptable RGB values
            output_img->color[0][i][j] = limitIntensity(redPlane);
            output_img->color[1][i][j] = limitIntensity(greenPlane);
            output_img->color[2][i][j] = limitIntensity(bluePlane);
        }
    }
}
```

4. FIR Sharpening Filter

1. Derivation of $H\left(e^{j\mu}, e^{j\nu}\right)$: *See Next Page*
2. Derivation of $G\left(e^{j\mu}, e^{j\nu}\right)$: *See Next Page*

Jack Girard
ECE637
Lab 1 Work

Let $h(m,n)$ be a low pass filter. For our purposes use

$$h(m,n) = \begin{cases} 1/25 & \text{for } |m| \le 2 \text{ and } |n| \le 2 \\ 0 & \text{otherwise} \end{cases}$$

The unsharp mask filter is then given by

$$g(m,n) = \delta(m,n) + \lambda(\delta(m,n) - h(m,n))$$

where λ is a constant greater than zero.

Identity Used:

$$sin(x) = \frac{e^{ix} - e^{-ix}}{2i}$$

$$cos(x) = \frac{e^{ix} + e^{-ix}}{2}$$

Apply formula for DSFT & Simplify:

$$H(e^{j\mu}, e^{j\nu}) = \sum_{n=-\infty}^{\infty} \sum_{m=-\infty}^{\infty} h(m,n) e^{-j(\mu m + \nu n)}$$

$$= \sum_{n=-2}^{2} \sum_{m=-2}^{2} \frac{1}{25} e^{-j(\mu m + \nu n)}$$

$$= \frac{1}{25} \sum_{n=-2}^{2} e^{-j\mu m} \sum_{m=-2}^{2} e^{-j\nu n}$$

$$= \frac{1}{25}\left[\left(1 + e^{-2\mu} + e^{2\mu} + e^{-\mu} + e^{\mu}\right)\left(1 + e^{-2\nu} + e^{2\nu} + e^{-\nu} + e^{\nu}\right)\right]$$

$$= \frac{1}{25}\left[\left(1 + 2cos(2\mu) + 2cos(\mu)\right)\left(1 + 2cos(2\nu) + 2cos(\nu)\right)\right]$$

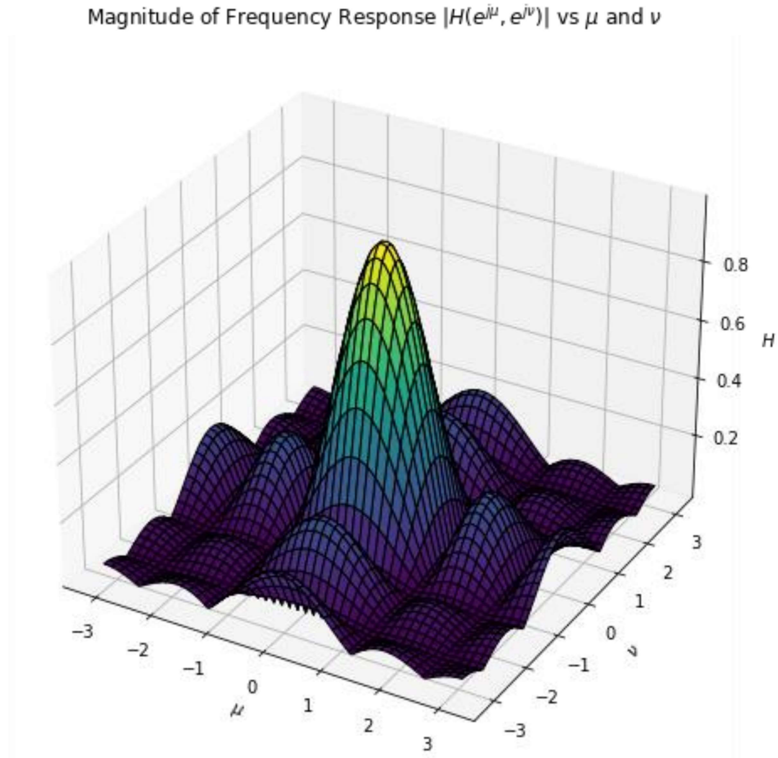
$$= \boxed{\frac{1}{25}\left[\left(1 + 2\sum_{k=1}^{2} cos(k\mu)\right)\left(1 + 2\sum_{l=1}^{2} cos(l\nu)\right)\right]}$$

Apply formula for DSFT & Simplify, knowing $H(e^{j\mu}, e^{j\nu})$:

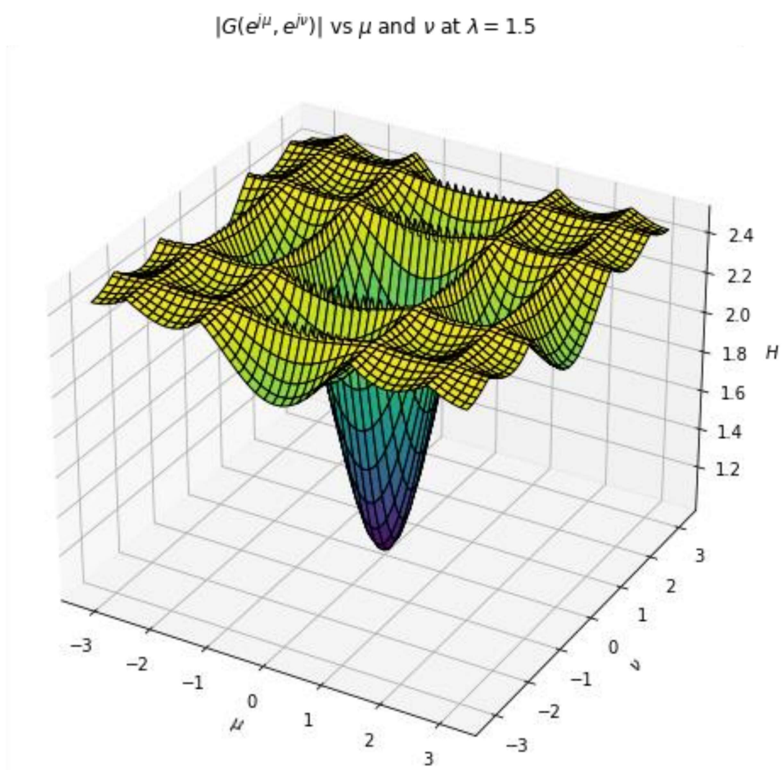
$$G(e^{j\mu}, e^{j\nu}) = \sum_{n=-\infty}^{\infty} \sum_{m=-\infty}^{\infty} g(m,n) e^{-j(\mu m + \nu n)} = 1 + \lambda(1 - H(e^{j\mu}, e^{j\nu}))$$

$$= \boxed{1 + \lambda\left[1 - \frac{1}{25}\left[\left(1 + 2\sum_{k=1}^{2} cos(k\mu)\right)\left(1 + 2\sum_{l=1}^{2} cos(l\nu)\right)\right]\right]}$$

3. Plot of $\left|H\left(e^{j\mu}, e^{j\nu}\right)\right|$



4. Plot of $\left|G\left(e^{j\mu}, e^{j\nu}\right)\right|$ for $\lambda = 1.5$



5. Input color *imgblur.tif*

6. Output sharpened color image, *sharpened.tif*, for $\lambda = 1.5$

7. Listing of C Code: *See Next Page*

```c

#include <math.h>
#include "tiff.h"
#include "allocate.h"
#include "randlib.h"
#include "typeutil.h"

void error(char *name);
// initialize limitIntensity function
int limitIntensity(double value);
// initialize applyFilter function
void applyFilter(struct TIFF_img* output_img, struct TIFF_img* input_img, double** usedFilter, int PSF_dim, double lambda);

int main (int argc, char **argv)
{
    FILE *fp;
    struct TIFF_img input_img, color_img;

    // adjustment: if number of arguments is greater than 3, since third will be lambda value
    if ( argc > 3 ) error( argv[0] );

    /* open image file */
    if ( ( fp = fopen ( argv[1], "rb" ) ) == NULL ) {
    fprintf ( stderr, "cannot open file %s\n", argv[1] );
    exit ( 1 );
    }

    /* read image */
    if ( read_TIFF ( fp, &input_img ) ) {
    fprintf ( stderr, "error reading file %s\n", argv[1] );
    exit ( 1 );
    }

    /* close image file */
    fclose ( fp );

    /* check the type of image data */
    if ( input_img.TIFF_type != 'c' ) {
    fprintf ( stderr, "error:  image must be 24-bit color\n" );
    exit ( 1 );
    }

    // declare 1D array of double pointers filter
    double* filter[5];
    // declare double to store scaling factor lambda - default value below
    double lambda = 1.0;
    if (argc > 2) {
```

```c
        lambda = atof(argv[2]);
    }

    // iterate through 1D array filter and allocate enough memory for 9 doubles each
    for (int i = 0; i < 5; i++) {
        filter[i] = malloc(sizeof(double) * 9);
    }
    // populate filter array according to g(m,n)
    for (int i = 0; i < 5; i++) {
        for (int j = 0; j < 5; j++) {
            if (i == 2 && j == 2) {
                filter[i][j] = 1 + lambda * (1 - 1.0 / 25.0);
            }
            else {
                filter[i][j] = lambda * (-1.0 / 25.0);
            }
        }
    }

    /* set up structure for output color image */
    /* Note that the type is 'c' rather than 'g' */
    get_TIFF ( &color_img, input_img.height, input_img.width, 'c' );

    // declare and initialize integer to store the dimension of the point spread function
    int PSF_dim = 5;

    // apply filter using applyFilter function as defined below main
    applyFilter(&color_img, &input_img, filter, PSF_dim, lambda);

    /* open color image file */
    if ( ( fp = fopen ( "sharpened.tif", "wb" ) ) == NULL ) {
        fprintf ( stderr, "cannot open file color.tif\n");
        exit ( 1 );
    }

    /* write color image */
    if ( write_TIFF ( fp, &color_img ) ) {
        fprintf ( stderr, "error writing TIFF file %s\n", argv[2] );
        exit ( 1 );
    }

    /* close color image file */
    fclose ( fp );

    /* de-allocate space which was used for the images */
    free_TIFF ( &(input_img) );
    free_TIFF ( &(color_img) );
```

```c

    return(0);
}

void error(char *name)
{
    printf("usage:  %s  image.tiff \n\n",name);
    printf("this program reads in a 24-bit color TIFF image.\n");
    printf("It then horizontally filters the green component, adds noise,\n");
    printf("and writes out the result as an 8-bit image\n");
    printf("with the name 'green.tiff'.\n");
    printf("It also generates an 8-bit color image,\n");
    printf("that swaps red and green components from the input image");
    exit(1);
}

// limitIntensity function definition
int limitIntensity(double inputValue) {
    // declare an integer variable newValue and initialize it to zero
    int newValue = 0;
    // if input value parameter is less than zero, assign new value to 0
    if (inputValue < 0) {
        newValue = 0;
    }
    // if input value parameter is greater than 255, assign new value to 255
    else if(inputValue > 255) {
        newValue = 255;
    }
    // otherwise, assign new value to the input value parameter re-cast as an integer
    else {
        newValue = (int)inputValue;
    }
    return newValue;
}

// applyFilter function definition
void applyFilter(struct TIFF_img* output_img, struct TIFF_img* input_img, double **usedFilter, int PSF_dim, double lambda) {
    // declare and define N - the dimension of the point spread function (PSF)
    int N = (PSF_dim - 1) / 2;
    // declare and define image height and width based on input image TIFF struct methods
    int img_height = input_img->height;
    int img_width = input_img->width;
    // declare doubles to store red, green, and blue value for each pixel
```

```c
    double redPlane, greenPlane, bluePlane;
    // declare PSF variables
    int m, n;
    // declare variables to store current location within PSF
    int a, b;
    // for each pixel
    for (int i = 0; i < img_height; i++) {
        for (int j = 0; j < img_width; j++) {
            // initialize RGB values to zero
            redPlane = 0.0;
            greenPlane = 0.0;
            bluePlane = 0.0;
            // for each pixel in the PSF (5*5 in this case)
            for (m = -N; m <= N; m++) {
                for (n = -N; n <= N; n++) {
                    // assign a and b to current PSF matrix location
                    a = i - m;
                    b = j - n;
                    // if a and b are within the image boundaries
                    if (a >= 0 && a < img_height && b >= 0 && b < img_width) {
                        // apply filter by summing across PSF according to difference equation for 2D filters
                        redPlane += usedFilter[m + N][n + N] * input_img->color[0][a][b];
                        greenPlane += usedFilter[m + N][n + N] * input_img->color[1][a][b];
                        bluePlane += usedFilter[m + N][n + N] * input_img->color[2][a][b];
                    }
                }

            }
            // populate output image method for color after calling limitIntensity function to ensure acceptable RGB values
            output_img->color[0][i][j] = limitIntensity(redPlane);
            output_img->color[1][i][j] = limitIntensity(greenPlane);
            output_img->color[2][i][j] = limitIntensity(bluePlane);
        }
    }
}
```

5. IIR Filter

1. Derivation of $H\left(e^{j\mu}, e^{j\nu}\right)$: *See Next Page*

Jack Girard
ECE637
Lab 1 Work

Let $h(m,n)$ be the impulse response of an IIR filter with corresponding difference equation

$$y(m,n) = 0.01x(m,n) + 0.9(y(m-1,n) + y(m,n-1)) - 0.81y(m-1,n-1)$$

where $x(m,n)$ is the input and $y(m,n)$ is the output.

Identity Used:

$$e^{i\phi} = \cos\phi + i\sin\phi$$

Taking Z-Transformation of difference equation:

$$Y(z_1,z_2) = 0.01X(z_1,z_2) + 0.9z_1^{-1}Y(z_1,z_2) + 0.9z_2^{-1}Y(z_1,z_2) - 0.81z_1^{-1}z_2^{-1}Y(z_1,z_2)$$

$$Y(z_1,z_2) - 0.9z_1^{-1}Y(z_1,z_2) - 0.9z_2^{-1}Y(z_1,z_2) + 0.81z_1^{-1}z_2^{-1}Y(z_1,z_2) = 0.01X(z_1,z_2)$$

$$Y(z_1,z_2)(1 - 0.9z_1^{-1} - 0.9z_2^{-1} + 0.81z_1^{-1}z_2^{-1}) = 0.01X(z_1,z_2)$$

$$H(z_1,z_2) = \frac{Y(z_1,z_2)}{X(z_1,z_2)} = \frac{0.01}{1 - 0.9z_1^{-1} - 0.9z_2^{-1} + 0.81z_1^{-1}z_2^{-1}}$$

$$H(e^{j\mu},e^{j\nu}) = \frac{0.01}{1 - 0.9e^{-j\mu} - 0.9e^{-j\nu} + 0.81e^{-j\mu}e^{-j\nu}}$$

$$\boxed{H(e^{j\mu},e^{j\nu}) = \frac{0.01}{1 - 0.9e^{-j\mu} - 0.9e^{-j\nu} + 0.81e^{-j(\mu+\nu)}}}$$

Further Simplifying $H(e^{j\mu},e^{j\nu})$ and calculating $|H(e^{j\mu},e^{j\nu})|$

$$= \frac{0.01}{1 - 0.9(\cos\mu - j\sin\mu) - 0.9(\cos\nu - j\sin\nu) + 0.81(\cos(\mu+\nu) - j\sin(\mu+\nu))}$$

$$= \frac{0.01}{[1 - 0.9\cos\mu - 0.9\cos\nu + 0.81\cos(\mu+\nu)] + [0.9j\sin\mu + 0.9j\sin\nu - 0.81j\sin(\mu+\nu)]}$$

$$= \frac{0.01}{\underbrace{[1 - 0.9(\cos\mu + \cos\nu) + 0.81\cos(\mu+\nu)]}_{=a} + j\underbrace{[0.9(\sin\mu + \sin\nu) - 0.81\sin(\mu+\nu)]}_{=b}}$$

$$= \frac{0.01}{a + jb} = \frac{0.01}{a + jb} \cdot \frac{a - jb}{a - jb} = \frac{0.01(a - jb)}{a^2 + b^2} = k(a - jb) \; ; \; k = \frac{0.01}{a^2 + b^2}$$

$$|k(a - jb)| = \sqrt{(ka)^2 + (kb)^2} = k\sqrt{a^2 + b^2} \; ; \; \begin{aligned} a &= 1 - 0.9(\cos\mu + \cos\nu) + 0.81\cos(\mu+\nu) \\ b &= 0.9(\sin\mu + \sin\nu) - 0.81\sin(\mu+\nu) \end{aligned}$$